Stream Processing in Python with Kafka

Austin Godber
@godber
DesertPy - 3/27/2019

DesertPy

Why stream processing?

- If it has a timestamp, it's a stream.
- Once your batch size gets large ...
  - you can't do real time processing.
  - stream processing may be cheaper, faster, and easier.
- Scaling to many consumers can be easier.

Overview

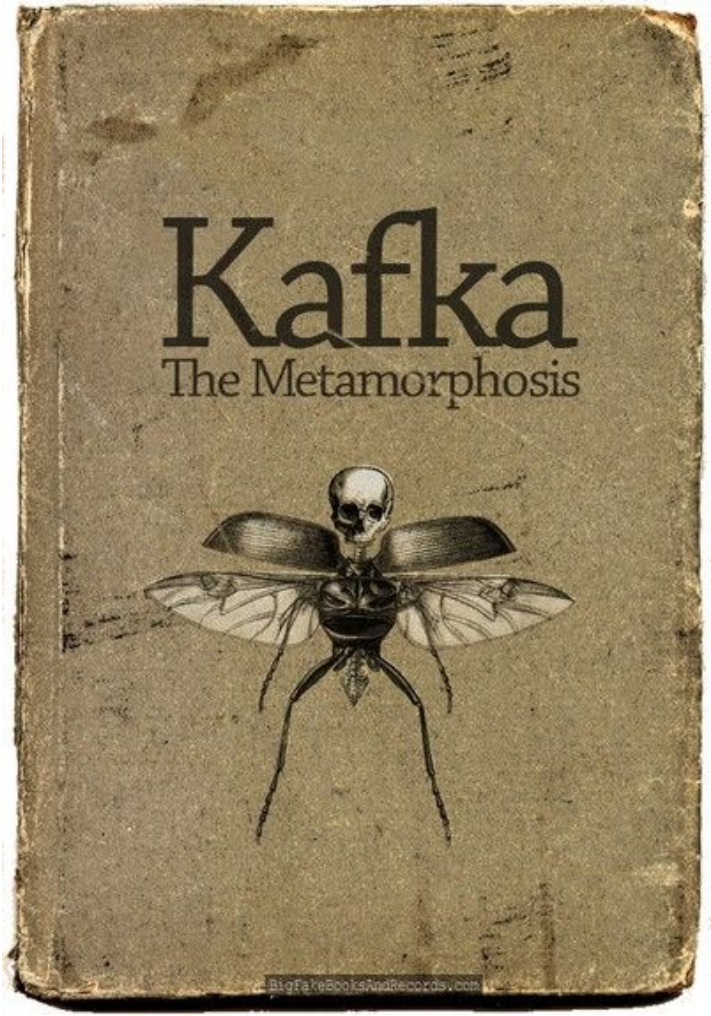
- Kafka Introduction
- Python and Kafka
- Stream Processing Problem

Kafka Introduction

What is Kafka?

A real-time streaming system.

- Distributed
- Persistent
- Resilient
- Fast



Kafka Capabilities

- Publish and subscribe to streams of records, similar to a message queue or enterprise messaging system.
- Store streams of records in a fault-tolerant durable way.
- Process streams of records as they occur.

Ref: https://kafka.apache.org/intro

Kafka Use Cases

- Building real-time streaming data pipelines that reliably get data between systems or applications
- Building real-time streaming applications that transform or react to the streams of data

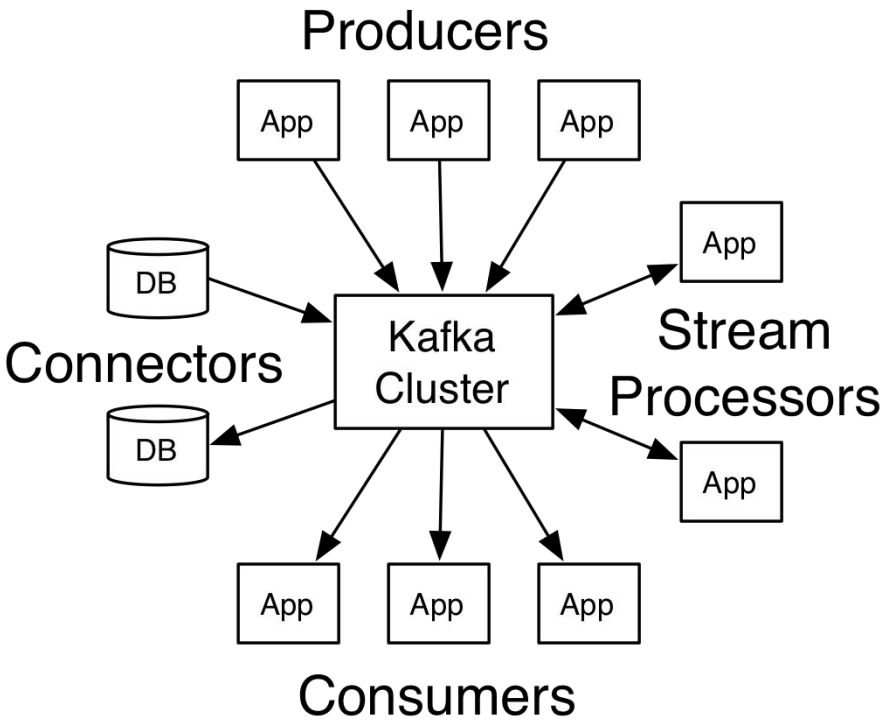
Ref: https://kafka.apache.org/intro

Kafka Concepts

- Kafka is run as a cluster on one or more servers that can span multiple datacenters.
- The Kafka cluster stores streams of records in categories called topics.
- Each record consists of a key, a value, and a timestamp.

Ref: https://kafka.apache.org/intro

Kafka High Level View

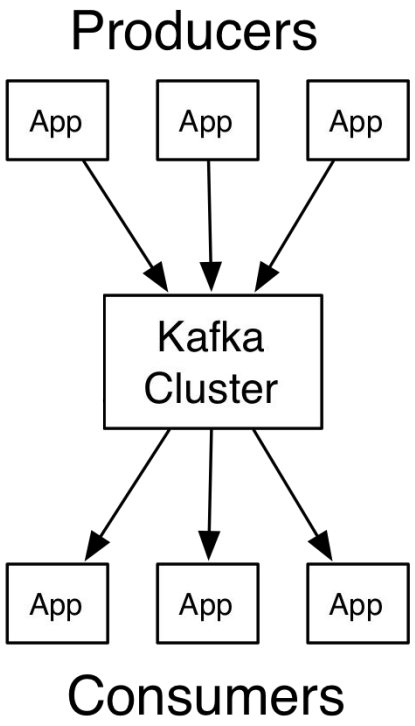


Ref: https://kafka.apache.org/intro

Kafka High Level View

I am going to ignore connectors and stream processors. These concepts were introduced and implemented later.

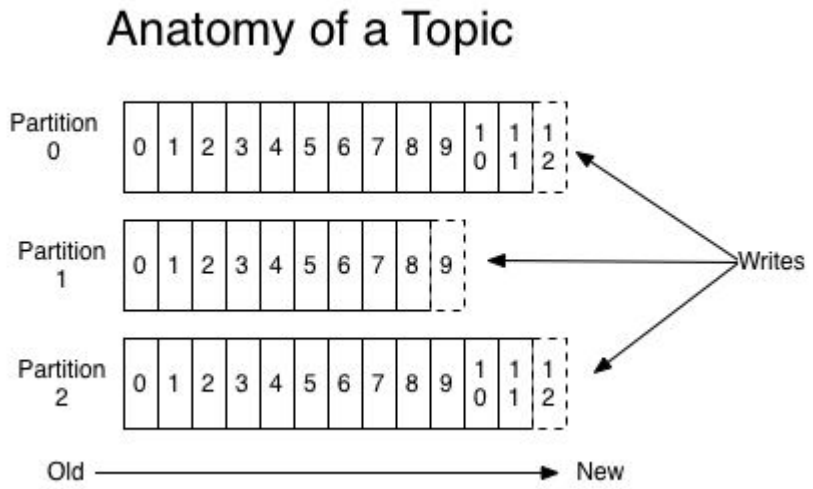
YOU may benefit from using them. Though there's no real Python tie-in.



Ref: https://kafka.apache.org/intro

Kafka Topics

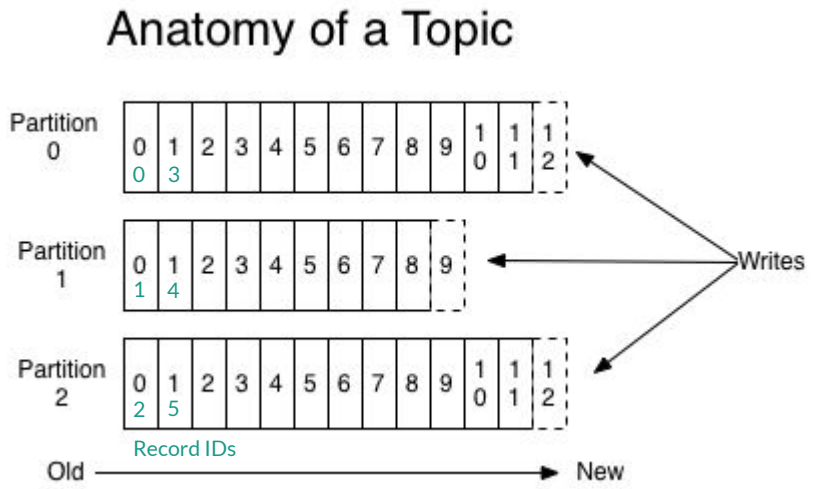
- Topics are split into append only logs called partitions.
- Each record has an `offset` within a partition
- Writes and reads are spread across partitions
- Topics have retention size/time can also be compacted



Ref: https://kafka.apache.org/intro

Kafka Topics

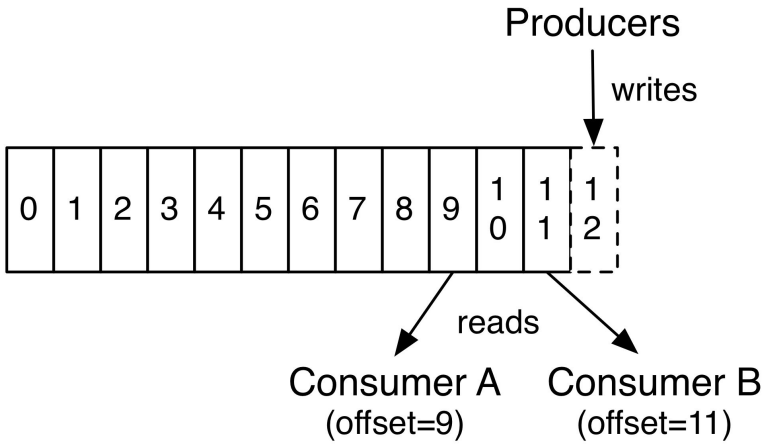
- Topics are split into append only logs called partitions.
- Each record has an `offset` within a partition
- Writes and reads are spread across partitions
- Topics have retention size/time can also be compacted



Ref: https://kafka.apache.org/intro

Kafka Consumers

- Multiple consumers can read from a topic
  - Each manages its own offsets
- Consumers identified by Consumer Group
- The number of partitions is the max parallelism for consumers in a given “Consumer Group”
- Consumers are very cheap (reading from the tail of the log)



Ref: https://kafka.apache.org/intro

Kafka Options for Python

Kafka and Python

There are two options for Kafka Libraries in Python

- `kafka-python` - designed to function much like the official java client, with a sprinkling of pythonic interfaces.
  - `https://kafka-python.readthedocs.io/`
- `aiokafka` - using asyncio, built on top of the kafka-python library.
  - `https://aiokafka.readthedocs.io/`

Basic Stream Program Flow

Consume -> Process -> Produce

Read the records from topic of interest

Implement your process, filter, enrich, change records here

Write the records out to a new topic, or new datastore, or tweet them

main function

Loops perpetually, executed with ...

```
loop.run_until_complete(main())
```

```
async def main():
    data_size = 10
    try:
        consumer = AIOKafkaConsumer(
            'noaa-json', loop=loop,
            bootstrap_servers='localhost:9092',
            group_id="e2-group-v1",
        )
        await consumer.start()

        producer = AIOKafkaProducer(
            loop=loop,
            bootstrap_servers='localhost:9092'
        )
        await producer.start()

        while True:
            data_array = await consume(consumer,data_size)
            await produce(producer, process(data_array))
    finally:
        await consumer.stop()
        await producer.stop()
```

main function

Loops perpetually, executed with ...

```
loop.run_until_complete(main())
```

```
async def main():
    data_size = 10
    try:
        consumer = AIOKafkaConsumer(
            'noaa-json', loop=loop,
            bootstrap_servers='localhost:9092',
            group_id="e2-group-v1",
        )
        await consumer.start()

        producer = AIOKafkaProducer(
            loop=loop,
            bootstrap_servers='localhost:9092'
        )
        await producer.start()

        while True:
            data_array = await consume(consumer,data_size)
            await produce(producer, process(data_array))
    finally:
        await consumer.stop()
        await producer.stop()
```

consume function

Reads over messages, does something, returns array of records

```
async def consume(consumer, data_array_size):
    records = []
    i = 0
    async for msg in consumer:
        i = i +1

        record = json.loads(msg.value)
        record['_id'] = msg.key

        records.append(record)

        if i > data_array_size:
            break

    return records
```

consume function

Reads over messages, does something, returns array of records

```
async def consume(consumer, data_array_size):
    records = []
    i = 0
    async for msg in consumer:
        i = i +1

        record = json.loads(msg.value)
        record['_id'] = msg.key

        records.append(record)

        if i > data_array_size:
            break

    return records
```

main function

Loops perpetually, executed with ...

```
loop.run_until_complete(main())
```

```
async def main():
    data_size = 10
    try:
        consumer = AIOKafkaConsumer(
            'noaa-json', loop=loop,
            bootstrap_servers='localhost:9092',
            group_id="e2-group-v1",
        )
        await consumer.start()

        producer = AIOKafkaProducer(
            loop=loop,
            bootstrap_servers='localhost:9092'
        )
        await producer.start()

        while True:
            data_array = await consume(consumer,data_size)
            await produce(producer, process(data_array))
    finally:
        await consumer.stop()
        await producer.stop()
```

process function

Modifies the records read from kafka, returns an array containing the modified records.

```
def process(data_array):
    out_data_array = []
    for record in data_array:
        if record['station']['country_code'] == 'US'
            and record['station']['state_code'] == 'AZ':
                out_data_array.append(record)
    return out_data_array
```

main function

Loops perpetually, executed with ...

```
loop.run_until_complete(main())
```

```
async def main():
    data_size = 10
    try:
        consumer = AIOKafkaConsumer(
            'noaa-json', loop=loop,
            bootstrap_servers='localhost:9092',
            group_id="e2-group-v1",
        )
        await consumer.start()

        producer = AIOKafkaProducer(
            loop=loop,
            bootstrap_servers='localhost:9092'
        )
        await producer.start()

        while True:
            data_array = await consume(consumer,data_size)
            await produce(producer, process(data_array))
    finally:
        await consumer.stop()
        await producer.stop()
```

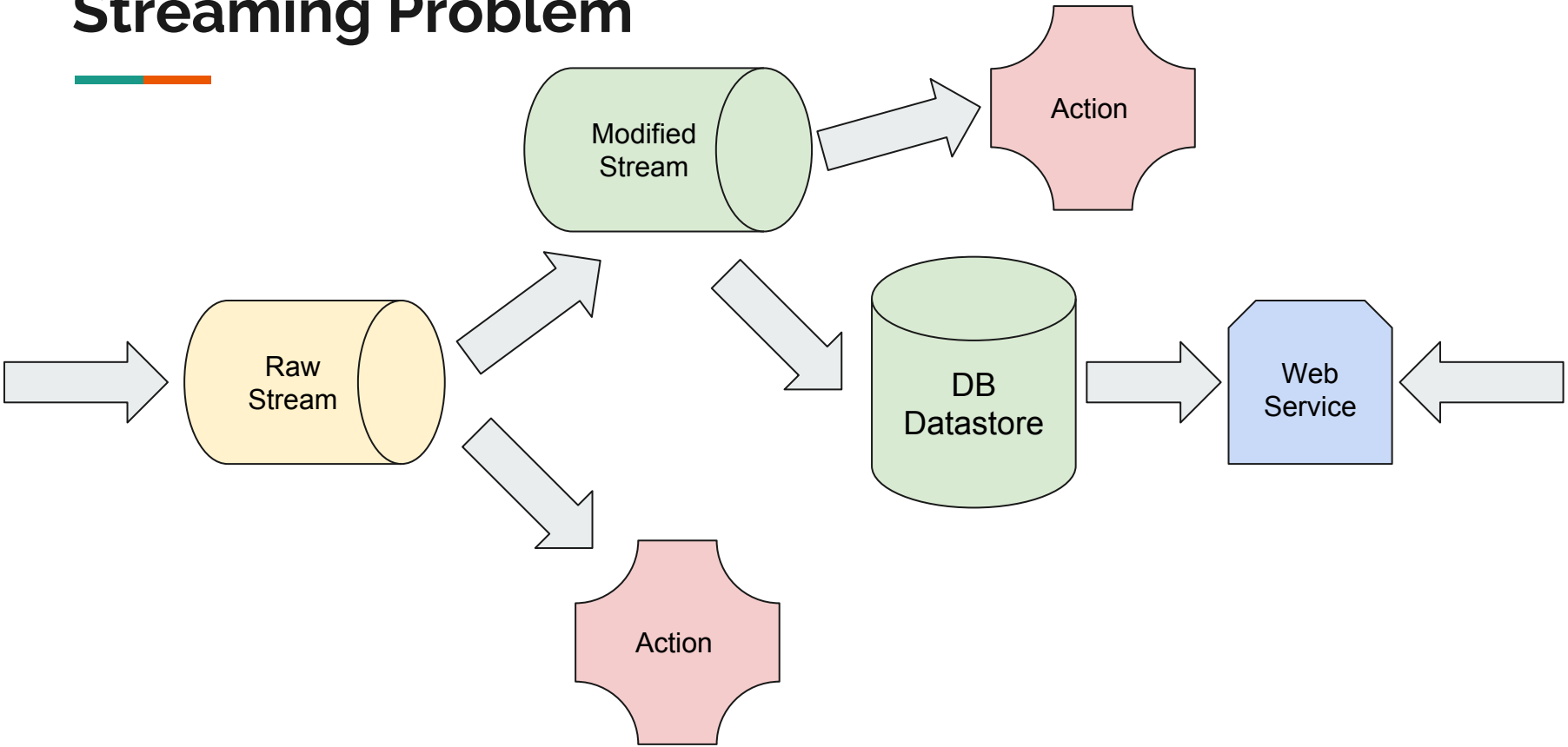
produce function

Loops over input messages, and writes them out

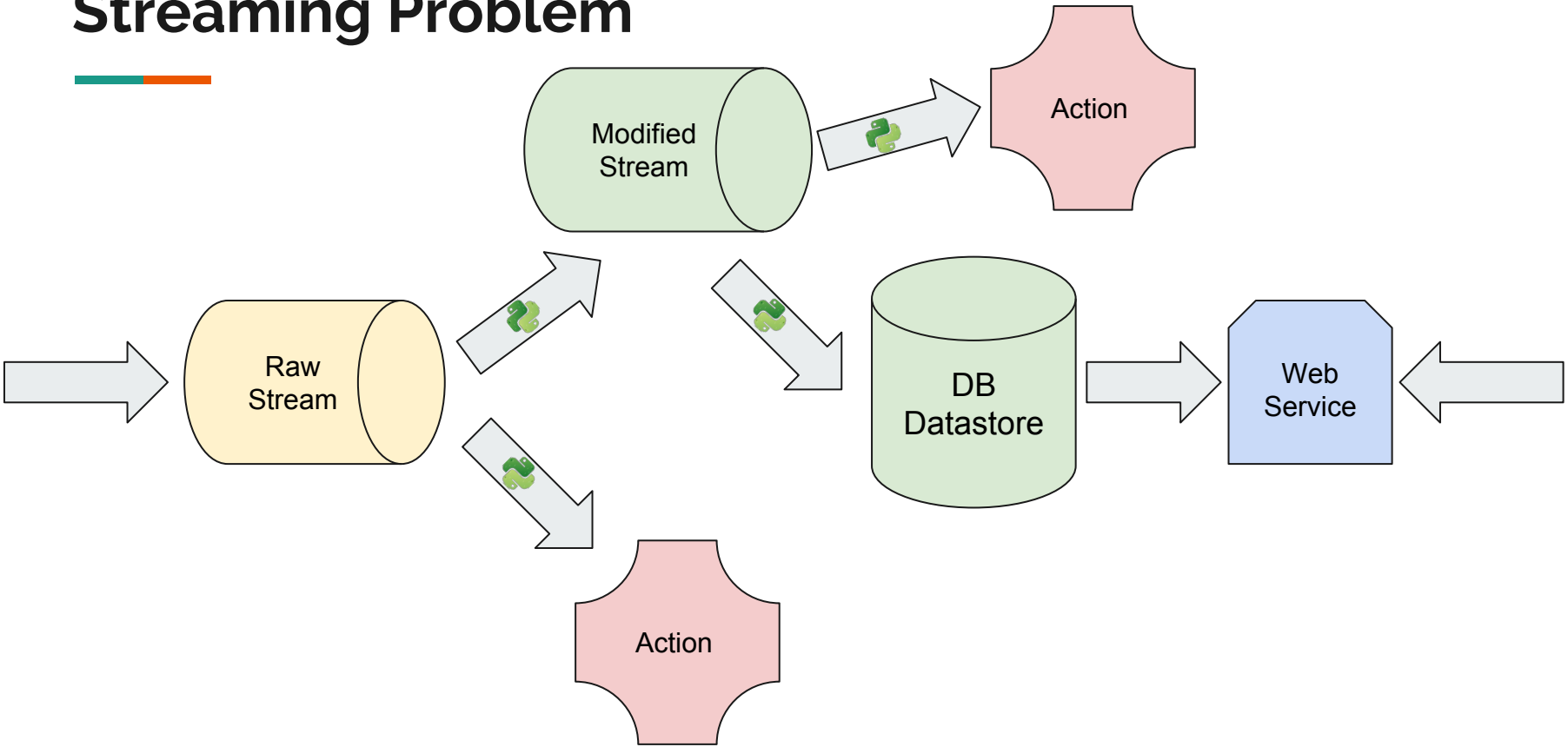
```
async def produce(producer, data_array):
    for record in data_array:
        await producer.send_and_wait(
              "noaa-json-us-az",
               json.dumps(record).encode('utf-8')
        )
```

Stream Processing Problem

Streaming Problem

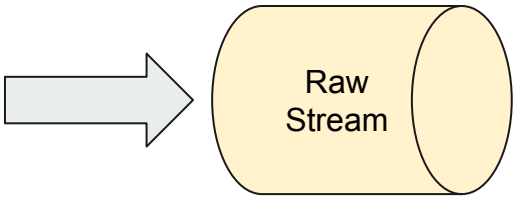


Streaming Problem



NOAA Streaming Problem

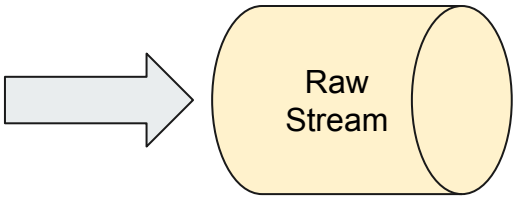
Incoming data is a big array of JSON records




```
{
    "station_id": "USW00003192",
    "date": "2016-04-23T00:00:00",
    "AWND": 2.4,
    "PRCP": 0,
    "TMAX": 29.4,
    "TMIN": 18.9,
}
```

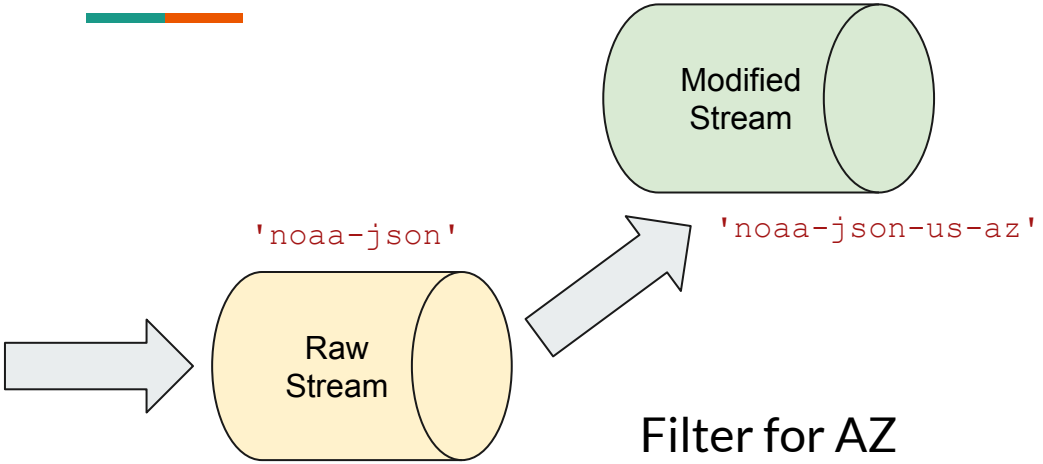
NOAA Streaming Problem

Incoming data is global and needs interpretation




```
{
    "station_id": "USW00003192",
    "date": "2016-04-23T00:00:00",
    "AWND": 2.4,
    "PRCP": 0,
    "TMAX": 29.4,
    "TMIN": 18.9,
}
```

NOAA Streaming Problem



Filter for AZ only and Data Enrichment

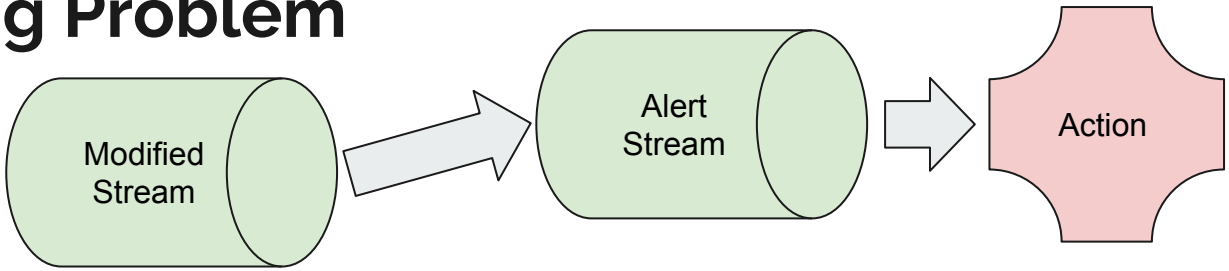

```
{
    "station_id": "USW00003192",
    "date": "2016-04-23T00:00:00",
    "AWND": 2.4,
    "PRCP": 0,
    "TMAX": 29.4,
    "TMIN": 18.9,
}
```

```
{
    "station": {
        "id": "USW00003192",
        "country_code": "US",
        "country": "United",
        "location": {
            "lat": 33.6228,
            "lon": -111.9106
        },
        "elevation": 449,
        "state_code": "AZ",
        "state": "ARIZONA",
        "name": "SCOTTSDALE MUNI AP"
    },
    "date": "2016-04-23T00:00:00",
    "AWND": 2.4,
    "PRCP": 0,
    "TMAX": 29.4,
    "TMIN": 18.9,
}
```

Say you had a farm near `USW00003192` and wanted an email, SMS or Notification if `TMIN` was approaching freezing.

NOAA Streaming Problem

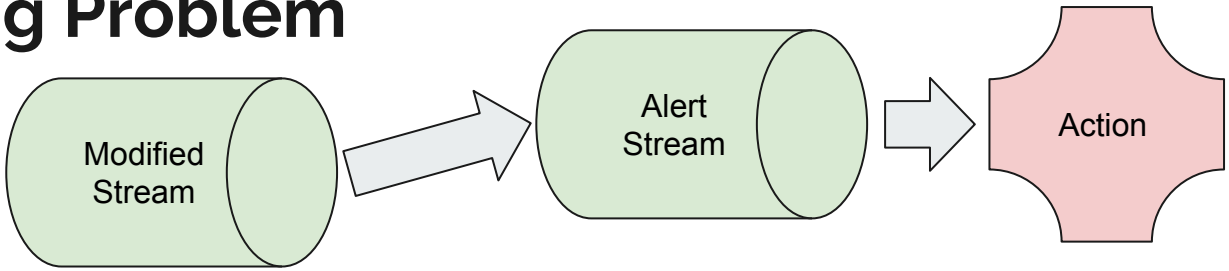

```
{
  "station": {
    "id": "USW00003192",
    "country_code": "US",
    "location": {
      "lat": 33.6228,
      "lon": -111.9106
    },
    "elevation": 449,
    "state_code": "AZ",
    "state": "ARIZONA",
    "name": "SCOTTSDALE MUNI AP"
  },
  "date": "2016-04-23T00:00:00",
  "AWND": 2.4,
  "PRCP": 0,
  "TMAX": 29.4,
  "TMIN": 18.9,
}
```

Filter for condition and specific station and alert user

NOAA Streaming Problem


```
{
  "station": {
    "id": "USW00003192",
    "country_code": "US",
    "location": {
      "lat": 33.6228,
      "lon": -111.9106
    },
    "elevation": 449,
    "state_code": "AZ",
    "state": "ARIZONA",
    "name": "SCOTTSDALE MUNI AP"
  },
  "date": "2016-04-23T00:00:00",
  "AWND": 2.4,
  "PRCP": 0,
  "TMAX": 29.4,
  "TMIN": 18.9,
}
```

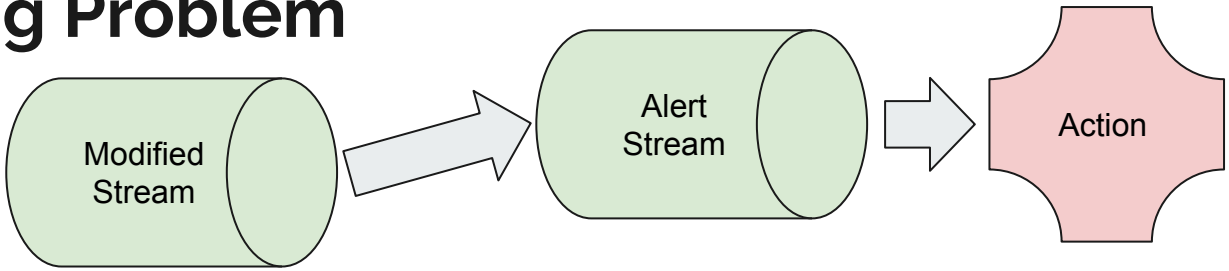



Filter for condition and specific station and alert user

```
# in main
consumer = AIOKafkaConsumer(
    'noaa-json-us-az',
    loop=loop,
    bootstrap_servers='localhost:9092',
    group_id="alert-group-v4",
    # earliest or latest
    auto_offset_reset="earliest"
)
```

NOAA Streaming Problem


```
{
  "station": {
    "id": "USW00003192",
    "country_code": "US",
    "location": {
      "lat": 33.6228,
      "lon": -111.9106
    },
    "elevation": 449,
    "state_code": "AZ",
    "state": "ARIZONA",
    "name": "SCOTTSDALE MUNI AP"
  },
  "date": "2016-04-23T00:00:00",
  "AWND": 2.4,
  "PRCP": 0,
  "TMAX": 29.4,
  "TMIN": 18.9,
}
```

Filter for condition and specific station and alert user

```
def process(data_array):
    alerts = [{
            'email': 'godber@gmail.com',
            'station_id': 'USW00003192'
        }]
    out_data_array = []
    for record in data_array:
        for alert in alerts:
            if (record['station']['id'] == alert['station_id']
                and record['TMIN'] <= 10.0):
                record['email'] = alert['email']
                out_data_array.append(record)
    return out_data_array
```

Why write to another topic rather than alerting immediately?

- You could, if the other topic serves no use and you can scale your alerting consumer enough to alert in a timely manner.
- Separating the conditional logic from the resulting action can help you scale.
- If you alert directly, you don't have a topic with all of the alerts that you can do other things with.
- It just depends on your needs. Topics aren't really expensive. Though tracking them and jobs can be.

Homework!

Try it out!

To try this code out yourself follow the instructions in the `README.md` (from GitHub) here:

http://bit.ly/2uAxnit

It should bootstrap your environment with Kafka and the fake streaming data and ability to stream it.

Note that without a **lot of RAM**, the data sorting steps might take a really long time. It went quickly on my Linux machine with 32 GB ram, but I have no idea whether it would have even finished on my 16 GB MacBook Pro.

References

- https://github.com/elastic/rally-tracks/tree/master/noaa
- https://aiokafka.readthedocs.io/en/stable/api.html

Thanks!

https://github.com/desertpy/presentations/

DesertPy

Backup

Streaming Problem

Show CSV Input

Use the difficulty of generating the JSON from multiline CSV as an example of how Kafka complicates things.

Get a solution … target partitions??

Streaming Problem